```python
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 18 17:38:08 2019

@author: Nate

The Generalized Metropolis algorithm removed the step-size error by an additional acceptance/rejection step,
which adds substantial overhead. To improve on the firstorder Langevin algorithm, can you devise a second-order
Langevin algorithm to reduce the step-size error dependence to (Δt)2?

Repeat problem 2 of HW10 using this second order Langevin algorithm.
"""

import numpy as np
import matplotlib.pyplot as plt
import pdb


###############################################################
###############################################################
#defining constants

N = 10000
alpha = 1.6875
g1,g2,g3,g4,g5,g6 = np.random.randn(N),np.random.randn(N),np.random.randn(N),np.random.randn(N),np.random.randn(N),np.random.randn(N)
x_int = np.array([4,2,3,-1,-4,-4])
del_t = np.append(np.arange(0.001,0.01,0.003), np.arange(0.01, 0.07, 0.01))
r_tot = np.zeros((len(del_t), N, len(x_int)))
en_dat = np.zeros((len(del_t), N))

###############################################################
###############################################################
#Function Definitions

def vel_func(r_tot):
    r1 = np.sqrt(np.sum(r_tot[:3]**2))
    r2 = np.sqrt(np.sum(r_tot[3:]**2))
    v1 = -alpha*r_tot[:3]/r1
    v2 = -alpha*r_tot[3:]/r2
    return np.append(v1,v2)

def lan(x_int, vel_func, gau, t, N):
    r_tot = np.zeros((N,len(x_int)))
    vel_func1 = vel_func(x_int)
    y0 = x_int + vel_func(x_int+t/4*vel_func1)*t/2 + gau[0]*np.sqrt(t)
    vel_func2 = vel_func(y0)
    r_tot[0] = y0 + t/2*vel_func(y0 + t/4*vel_func2)
    
    for i in range(1,N):
        vel_func1 = vel_func(r_tot[i-1])
        Yi = r_tot[i-1] + vel_func(r_tot[i-1] + t/4*vel_func1)*t/2 + gau[i]*np.sqrt(t)
        vel_func2 = vel_func(Yi)
        r_tot[i] = Yi + t/2*vel_func(Yi + t/4*vel_func2)
    return r_tot

def en(x1, y1, z1, x2, y2, z2, alpha):
    
    r1 = np.sqrt(x1**2 + y1**2 + z1**2)
    r2 = np.sqrt(x2**2 + y2**2 + z2**2)
    r_diff = np.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)
    
    return alpha * (-alpha + 1/r1 + 1/r2) - 2/r1 - 2/r2 + 1/r_diff

###############################################################
###############################################################
```

```
#Main Loop

for i in range(len(del_t)):
    
    r_tot[i] = lan(x_int, vel_func, gau, del_t[i], N)
    
    x1 = np.array([r_tot[i,j,0] for j in range(N)])
    y1 = np.array([r_tot[i,j,1] for j in range(N)])
    z1 = np.array([r_tot[i,j,2] for j in range(N)])
    x2 = np.array([r_tot[i,j,3] for j in range(N)])
    y2 = np.array([r_tot[i,j,4] for j in range(N)])
    z2 = np.array([r_tot[i,j,5] for j in range(N)])
    
    en_dat[i] = en(x1, y1, z1, x2, y2, z2)
    
en_arr = np.average(en_dat, axis=1)
err2 = np.std(en_dat, axis=1)/np.sqrt(N/48**2)

##############################################################
##############################################################
#Plotting

fig1, axes1 = plt.subplots()
axes1.plot(del_t, en_arr, 'o', label = '2nd Order Langevin')
axes1.hlines(-729/256, 0, np.max(del_t), linestyle='dashed', label = 'Theoretical')
axes1.set_ylabel('Energy')
axes1.set_xlabel('Time Step Sizes $\Delta t$')
axes1.set_title("Energy vs $\Delta t$", va='bottom')
axes1.legend()
plt.show()


#####################################################
#####################################################
```

For problem 3, I used the fortran code provided, adding only a 4th order Forest Ruth:

```
!-----------subprograms-----------------------------------------------
      Subroutine schem4A(m,N,ExpTA,ExpVA,psi,ExpThalfA,ExpVhalfA,ExpTfullA,ExpVfullA)
! 
!   To calculate the 4th-order decomposition scheme.
! 
      parameter (Ndim=16384/2)
      complex*16 ExpTA(1),ExpVA(1),psi(1),ExpThalfA(1),ExpVhalfA(1),ExpTfullA(1),ExpVfullA(1),phi(Ndim)
!
        DO i = 1, N
          phi(i) = ExpVhalfA(i) * psi(i)
        END DO
        call fft (phi, m, 0)
        DO i = 1, N
          phi(i) = ExpThalfA(i) * phi(i)
        END DO
        call fft (phi, m, 1 )
        DO i = 1, N
          psi(i) = ExpVhalfA(i) * phi(i)
        END DO
        DO i = 1, N
          phi(i) = ExpVfullA(i) * psi(i)
        END DO
        call fft (phi, m, 0)
        DO i = 1, N
          phi(i) = ExpTfullA(i) * phi(i)
        END DO
        call fft (phi, m, 1 )
        DO i = 1, N
          psi(i) = ExpVfullA(i) * phi(i)
        END DO
```

```
        DO i = 1, N
          phi(i) = ExpVhalfA(i) * psi(i)
        END DO
        call fft (phi, m, 0)
        DO i = 1, N
          phi(i) = ExpThalfA(i) * phi(i)
        END DO
        call fft (phi, m, 1 )
        DO i = 1, N
          psi(i) = ExpVhalfA(i) * phi(i)
        END DO
!*
       return
       end
```